

Anotações dos meus cursos



Não confie em sua memória, ela é uma rede cheia de buracos, os mais belos prêmios escorregam através deles.

(Georges Duhamel)

kdfrases

Este documento contém as anotações de cursos e materiais que não são de minha autoria. Como a memória é falha e os pontos de vista mudam com o passar do tempo, precisamos nos acostumar a ler e reler o que nos foi ensinado, para ter uma compreensão melhor das coisas.

Cleusa Granatto

Sumário

Curso: Introdução ao git e ao github.....	3
Comandos cmd / prompt.....	3
Chave ssh e tokens	4
Ciclo de vida dos arquivos no git.....	7
Inicializando o repositório do Git	7
Fluxo de trabalho do Git	7
Entendendo o fluxo do git	7
Configurações da conta do usuário	8
Verificando as informações do usuário.....	8
Listando todas as informações	8
Versão do git.....	8
Configurando usuário	8
Corrigindo as informações do usuário	8
Preparando o commit.....	8
Criando um repositório na página do git.....	9
Fazendo o push.....	10
Resolvendo conflitos.....	11
Comandos básicos do git.....	12
Curso: Dominando IDEs	18
Rodando um programa no terminal, sem o auxílio de uma IDE.	18

Curso: Introdução ao git e ao github

Anotações do curso de git e github oferecido pela DIO. Também estão presentes neste documento alguns materiais localizados na internet.

Comandos cmd / prompt

Windows	Linux	Descrição
dir	ls	Lista diretórios
cd /	cd	Navegar por diretórios
cd..	cd..	Retrocede um nível
cls	Clear ou ctrl + l	Limpar tela
não tem	tab	Auto completar
mkdir	mkdir	Criar pastas
rm -rf	rm -rf	Remove diretório
echo hello > hello.txt	echo hello > hello.txt	Cria um documento txt
	sudo su	Acesso como root
	pwd	Mostra caminho
	ls -a	Mostra arquivos ocultos
	mv arquivo ./pasta/	Mover arquivo para pasta
	q	sair

Para ativar o Linux no Windows

<https://www.youtube.com/watch?v=HSLpH43hL-o>

Ou no PowerShell digite:

Enable-WindowsOptionalFeature-Online-FeatureNameMicrosoft-Windows-Subsystem-Linux

Iniciando o uso do git

O que é um repositório Git?

Um repositório do Git é um armazenamento virtual para projetos. Ele permite salvar versões do código, que podem ser acessadas quando precisar.

Se já tiver uma pasta do projeto basta entrar no cmd e localizá-la. Em seguida, criar um arquivo Readme para descrever o conteúdo da pasta. Pode ser Readme.txt ou Readme.md (normalmente o mais utilizado).

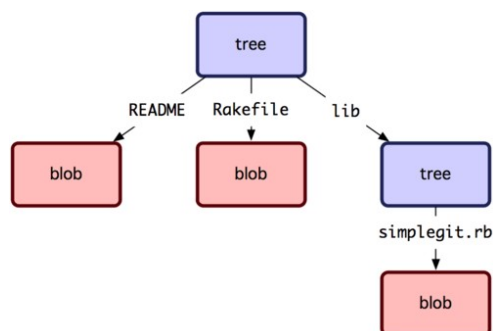
Usando o git bash here

Por exemplo:

Botão direito na tela Desktop > Git Bash Here -> abre o git na pasta Desktop.

openssl sha1 texto.txt

Elementos do git



A cada alteração de arquivo ocorre alteração em toda a estrutura da árvore.

Chave ssh e tokens


Chave SSH: forma de estabelecer uma conexão segura e encriptada entre duas máquinas.

Token de acesso pessoal: criar um token pessoal, salvar em um local seguro e acessar o git usando http. Não é tão eficiente quanto a chave SSH.

Gerando a chave SSH

1. No Windows, abrir o **Git Bash** e digitar: `ssh-keygen -t ed25519 -C email_github`
2. Aperte enter para confirmar o local de instalação e coloque uma senha.
3. Para visualizar a pasta das chaves digitar: `cd /c/Users/Nome_Usuario/.ssh/`
4. Para acessar o arquivo da chave digite: `Cat id_ed25519.pub` (chave pública)
5. Copiar o código da chave pública e criar a chave no SSH keys (site do git).
6. Voltando ao Git Bash, inicialize o SSH Agent, que é uma entidade que vai trabalhar com as chaves.
7. Em seguida, adicionar a chave com: `ssh-add id_ed25519`. Como está na pasta correta, é só digitar o nome da chave.

Após o passo 1, siga o passo 2.



A terminal window titled "MINGW64:/c/Users/Cleusa" with standard window controls. The prompt is "Cleusa@DESKTOP-VS39B1B MINGW64 ~". The command "\$ ssh-keygen -t ed25519 -C cleusafgranatto@gmail.com" has been entered. The output shows "Generating public/private ed25519 key pair." and the prompt "Enter file in which to save the key (/c/Users/Cleusa/.ssh/id_ed25519): |".

```
MINGW64:/c/Users/Cleusa
Cleusa@DESKTOP-VS39B1B MINGW64 ~
$ ssh-keygen -t ed25519 -C cleusafgranatto@gmail.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Cleusa/.ssh/id_ed25519): |
```

Confirmação da criação das chaves.

```

MINGW64:/c/Users/Cleusa
$ ssh-keygen -t ed25519 -C cleusafgranatto@gmail.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Cleusa/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Cleusa/.ssh/id_ed25519
Your public key has been saved in /c/Users/Cleusa/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:UbtnlbTbfhhY6rEveh0jFq8RUAIfN8kTy2d694MNuEO cleusafgranatto@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|      ..+o=o.      |
|      o.*++ o      |
|      . o.o.B       |
|      . .+O o       |
|      S ..BE+ o     |
|      +B+B=.        |
|      ooB.=+        |
|      +.. o         |
|      .o ..         |
+-----[SHA256]-----+

Cleusa@DESKTOP-VS39B1B MINGW64 ~
$

```

Passos 3 e 4.



```
MINGW64:/c/Users/Cleusa/.ssh

Cleusa@DESKTOP-VS3981B MINGW64 ~/.ssh
$ cd /c/Users/Cleusa/.ssh/

Cleusa@DESKTOP-VS3981B MINGW64 ~/.ssh
$ ls
id_ed25519  id_ed25519.pub

Cleusa@DESKTOP-VS3981B MINGW64 ~/.ssh
$ cat id_ed25519.pub
ssh-ed25519 [REDACTED]
cleusafrgratto@gmail.com

Cleusa@DESKTOP-VS3981B MINGW64 ~/.ssh
$ |
```

Copiar

Na página do GitHub cole a chave copiada no passo anterior.

SSH keys / Add new

Title

minha máquina Windows

Identifique

Key

ssh-ed25519 [redacted]
cleusafrgranatto@gmail.com

Copiar

Add SSH key

Startando o processo

```
Cleusa@DESKTOP-VS39B1B MINGW64 ~/.ssh
$ pwd
/c/Users/Cleusa/.ssh

Cleusa@DESKTOP-VS39B1B MINGW64 ~/.ssh
$ eval $(ssh-agent -s)
Agent pid 1206

Cleusa@DESKTOP-VS39B1B MINGW64 ~/.ssh
$ ssh-add id_ed25519
Enter passphrase for id_ed25519:
Identity added: id_ed25519 (cleusafrgranatto@gmail.com)

Cleusa@DESKTOP-VS39B1B MINGW64 ~/.ssh
$ |
```

Digitar

Digitar

Ciclo de vida dos arquivos no git

Inicializando o repositório do Git

Inicializa o repositório do Git dentro de uma pasta.

- Entre na pasta escolhida usando o **git bash here** para fazer o versionamento e no terminal digite:

`git init`

Fluxo de trabalho do Git



Antes de executar o commit



Ao executar o commit

Entendendo o fluxo do git

A partir daqui, vamos entender o fluxo do Git.

Working directory: é onde estão seus arquivos que serão trabalhados. Aqui ficam os arquivos que ainda não receberam o comando `commit`.

Index ou Staging area: é a sala de espera do Git. É aqui que ficam os arquivos que já receberam o comando `commit`.

Repositório: é onde são guardados os seus commits.

Configurações da conta do usuário

Verificando as informações do usuário

Um dos comandos git mais usados é o git config que pode ser usado para definir valores de configuração específicos do usuário como e-mail, algoritmo preferido para diff, nome de usuário e formato de arquivo etc.

Listando todas as informações

```
git config --list
```

Versão do git

Abra o cmd e digite: git --version

Imprime a versão do pacote Git de onde veio o programa git .

Configurando usuário

```
git config --global user.name "nome"
```

```
git config --global user.nickname "nome"
```

```
git config --global user.email "email"
```

```
git config --global core.editor nomeEditor
```

Corrigindo as informações do usuário

Para corrigir basta digitar: **--unset** e adicionar a configuração correta. Por exemplo:

```
git config --global --unset user.nickname (exclusão).
```

```
git config --global user.nickname "novonickname" (correção).
```

Preparando o commit

Adicionando arquivos

```
git add <nome_do_arquivo> (Move os arquivos para o Staged).
```

```
git add * (Move todos os arquivos modificados para o Staged).
```

Fazendo o commit de duas maneiras

```
git commit -m "Primeiro Commit" (commit → adicionar os arquivos antes com o git add)
```

```
git commit -am "Primeiro Commit" (Adicionar os arquivos e commitar)
```



```

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ ls
'Introdução ao Git e ao GitHub'/  README.md

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  "Introdu\303\247\303\243o ao Git e ao GitHub/"
  README.md

nothing added to commit but untracked files present (use "git add" to track)

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ git add *

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ git commit -m "Primeiro Commit"
[master (root-commit) 554c38b] Primeiro Commit
 2 files changed, 10 insertions(+)
 create mode 100644 "Introdu\303\247\303\243o ao Git e ao GitHub/Anota\303\247\3
03\265es_git_github.pdf"
 create mode 100644 README.md

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ |

```

Criando um repositório na página do git

Na página do git, criar um repositório (público ou privado).



No bash vamos apontar para o repositório criado no git.

git remote add origin **endereço da pasta do repositório**

```
Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-git
hub-primeiro-repositorio (master)
$ git remote add origin git@github.com:Granatto/dio-desafio-github-primeiro-repositorio.git

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ |
```

git remote -v

```
Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ git remote -v
origin  git@github.com:Granatto/dio-desafio-github-primeiro-repositorio.git (fetch)
origin  git@github.com:Granatto/dio-desafio-github-primeiro-repositorio.git (push)

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ git status
On branch master
nothing to commit, working tree clean

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ |
```

Fazendo o push

Depois de adicionar os arquivos em staged, faça o commit e verifique com o git status. Se não houver nenhuma pendência, digite o comando git push origin main para enviar os arquivos para o repositório remoto criado no git.

```
Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ git status
On branch master
nothing to commit, working tree clean

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ git push origin master
The authenticity of host 'github.com (20.201.28.151)' can't be established.
ED25519 key fingerprint is SHA256:++
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enter passphrase for key '/c/Users/Cleusa/.ssh/id_ed25519':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.63 MiB | 1.27 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Granatto/dio-desafio-github-primeiro-repositorio.git
 * [new branch]      master -> master

Cleusa@DESKTOP-VS39B1B MINGW64 ~/Documents/MEGAsync/04. Projetos/dio-desafio-github-primeiro-repositorio (master)
$ |
```

Resolvendo conflitos

Quando o repositório do git(main) é alterado, é necessário puxar os arquivos alterados, resolver os conflitos na pasta local (em seu computador) e somente depois, enviar novamente para o repositório remoto.

git pull origin main (puxando os arquivos do git main).

git push origin main (empurrando os arquivos já alterados para o git main).

Comandos básicos do git

- **git config**

Um dos comandos git mais usados é o **git config** que pode ser usado para definir valores de configuração específicos do usuário como e-mail, algoritmo preferido para diff, nome de usuário e formato de arquivo etc. Por exemplo, o seguinte comando pode ser usado para definir o email:

```
git config --global user.email sam@google.com
```

- **git init**

Este comando é usado para criar um novo repositório GIT. Uso:

```
git init
```

- **git add**

O comando **git add** pode ser usado para adicionar arquivos ao índice. Por exemplo, o seguinte comando irá adicionar um arquivo chamado temp.txt presente no diretório local para o índice:

```
git add temp.txt
```

- **git clone**

O comando **git clone** é usado para fins de verificação de repositório. Se o repositório estiver em um servidor remoto, use:

```
git clone alex@93.188.160.58:/path/to/repository
```

Por outro lado, se uma cópia de trabalho de um repositório local for criada, use:

```
git clone /path/to/repository
```

- **git commit**

O comando **git commit** é usado para confirmar as alterações na cabeça. Tenha em atenção que quaisquer alterações efetuadas não irão para o repositório remoto. Uso:

```
git commit -m "coloque sua mensagem aqui"
```

- **git status**

O comando **git status** exibe a lista de arquivos alterados juntamente com os arquivos que ainda não foram adicionados ou confirmados. Uso:

```
git status
```

git push é outro dos comandos git básicos mais usados. Um simples envio envia as alterações feitas para o ramo mestre do repositório remoto associado ao diretório de trabalho. Por exemplo:

```
git push origin master
```

- **git checkout**

O comando **git checkout** pode ser usado para criar ramos ou alternar entre eles. Por exemplo, o seguinte cria um novo ramo e muda para ele:

```
command git checkout -b <branch-name>
```

Para simplesmente mudar de um ramo para outro, use:

```
git checkout <branch-name>
```

- **git remote**

O comando **git remote** permite que um usuário se conecte a um repositório remoto. O comando a seguir lista os repositórios remotos atualmente configurados:

```
git remote -v
```

Esse comando permite que o usuário se conecte a um servidor remoto:

```
git remote add origin <93.188.160.58>
```

- **git branch**

O comando **git branch** pode ser usado para listar, criar ou excluir ramos. Para listar todos os ramos presentes no repositório, use:

```
git branch
```


Para excluir um ramo:

```
git branch -d <branch-name>
```

- **git pull**

Para mesclar todas as alterações presentes no repositório remoto para o diretório de trabalho local, o comando pull é usado. Uso:

```
git pull
```

- **git merge**

O comando **git merge** é usado para mesclar uma ramificação no ramo ativo. Uso:

```
git merge <branch-name>
```

- **git diff**

O comando **git diff** é usado para listar os conflitos. Para visualizar conflitos com o arquivo base, use

```
git diff --base <file-name>
```

O seguinte comando é usado para exibir os conflitos entre ramos about-to-be-merged antes de mesclá-los:

```
git diff <source-branch> <target-branch>
```

Para simplesmente listar todos os conflitos atuais, use:

```
git diff
```

- **git tag**

A marcação é usada para marcar compromissos específicos com alças simples. Um exemplo pode ser:

```
git tag 1.1.0 <insert-commitID-here>
```

- **git log**

Executar o comando **git log** exibe uma lista de compromissos em uma ramificação, juntamente com os detalhes pertinentes. Um exemplo de saída pode ser:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadm
Author: Alex Hunter <alexh@gmail.com>
Date:   Mon Oct 1 12:56:29 2016 -0600
```

- **git reset**

Para redefinir o índice e o diretório de trabalho para o estado do último commit, o comando **git reset** é usado. Uso:

```
git reset --hard HEAD
```

- **git rm**

git rm pode ser usado para remover arquivos do índice e do diretório de trabalho. Uso:

```
git rm filename.txt
```

- **git stash**

Provavelmente um dos menos conhecidos comandos git básicos é **git stash** que ajuda a salvar as mudanças que não devem ser cometidos imediatamente, mas em uma base temporária. Uso:

```
git stash
```

- **git show**

Para visualizar informações sobre qualquer objeto git, use o comando **git show**. Por exemplo:

```
git show
```

- **git fetch**

git fetch permite que um usuário obtenha todos os objetos do repositório remoto que atualmente não residem no diretório de trabalho local. Exemplo de uso:

```
git fetch origin
```

- **git ls-tree**

Para exibir um objeto de árvore juntamente com o nome e o modo de cada item e o valor SHA-1 do blob, use o comando **git ls-tree**. Por exemplo:

```
git ls-tree HEAD
```

- **git cat-file**

Usando o valor SHA-1, exiba o tipo de um objeto usando o comando **git cat-file**. Por exemplo:

```
git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

- **git grep**

git grep permite que um usuário procure através das árvores de conteúdo frases e / ou palavras. Por exemplo, para pesquisar `www.hostinger.com` em todos os arquivos use:

```
git grep "www.hostinger.com"
```

- **gitk**

gitk é a interface gráfica para um repositório local que pode ser invocado digitando e executando:

`gitk`

- **git instaweb**

Com o comando **git instaweb**, um servidor web pode ser executado em interface com o repositório local. Um navegador da Web também é automaticamente direcionado para ele. Por exemplo:

```
git instaweb -httpd=webrick
```

- **git gc**

Para otimizar o repositório através da coleta de lixo, que irá limpar arquivos desnecessários e otimizá-los, use:

```
git gc
```


- **git archive**

O comando **git archive** permite que um usuário crie um arquivo zip ou tar contendo os componentes de uma única árvore de repositório. Por exemplo:

```
git archive --format=tar master
```

- **git prune**

Através do comando **git prune**, os objetos que não têm ponteiros de entrada são excluídos. Uso:

```
git prune
```

- **git fsck**

Para executar uma verificação de integridade do sistema de arquivos git, use o comando **git fsck**. Todos os objetos corrompidos são identificados:

```
git fsck
```

- **git rebase**

O comando **git rebase** é usado para reaplicação de compromissos em outro ramo. Por exemplo:

```
git rebase master
```

Fonte: Hostinger Tutoriais

https://www.hostinger.com.br/tutoriais/comandos-basicos-degit?ppc_campaign=google_performance_max&gclid=EAlaIQobChMlvOm49u2B9QIVDoCRCh35pQDVEAAYASAAEgIN-vD_BwE

Curso: Dominando IDEs

Rodando um programa no terminal, sem o auxílio de uma IDE.

1. Criar o programa abaixo usando o bloco de notas

```
public class PrimeiroPrograma {  
    public static void main(String args[]){  
        System.out.println("Hello World!");  
    }  
}
```

2. Indicar o local do arquivo a ser compilado

3. Compilar o arquivo com o comando **javac**

```
javac PrimeiroPrograma.java
```

4. Executar o arquivo

```
java PrimeiroPrograma
```